CHAPTER 12

MongoDB Best Practices

"Getting started with MongoDB is easy, but once you start developing applications you will come across scenarios where you may need best practices to achieve particular use cases."

In the previous chapters, you became acquainted with MongoDB. The intent of this chapter is outline well-known issues using other user's experiences but also to provide various How To's that can help your journey with MongoDB be a smooth ride.

As you know, MongoDB works with documents, uses RAM for storing data to enhance performance, and uses replication and sharding to further provide data safety and scalability.

This chapter will cover tips that you should be aware of, from the deployment strategy to enhancing querying to data safety and consistency to monitoring.

Deployment

While deciding on the deployment strategy, keep the following tips in mind so that the hardware sizing is done appropriately. These tips will also help you decide whether to use sharding and replication.

- **Data set size**: The most important thing is to determine the current and anticipated data set size. This not only lets you choose resources for individual physical nodes, but it also helps when planning your sharding plans (if any).
- **Data importance**: The second most important thing is to determine data importance, to determine how important the data is and how tolerant you can be to any data loss or data lagging (especially in case of replication).
- **Memory sizing**: The next step is to identify memory needs and accordingly take care of the RAM.

  Like other data-oriented applications, MongoDB also works best when the entire data set can reside in memory, thereby avoiding any kind of disk I/O.

  Page faults indicate that you may exceed the available deployment's memory and should consider increasing it. Page fault is a metric that can be measured using monitoring tools like MongoDB Cloud Manager.

  If possible, you should always select a platform that has memory greater than your working set size.

  If the size exceeds the single node's memory, you should consider using sharding so that the amount of available memory can be increased. This maximizes the overall deployment's performance.

S.G. Edward and N. Sabharwal, *Practical MongoDB*, DOI 10.1007/978-1-4842-0647-8_12

- **Disk Type**: If speed is not a primary concern or if the data set is larger than what any in-memory strategy can support, it's very important to select a proper disk type. IOPS (input/output operations per second) is the key for selecting a disk type; the higher the IOPS, the better the MongoDB performance. If possible, local disks should be used because network storage can cause poor performance and high latency. It is also advised to use RAID 10 when creating disk arrays (wherever possible).
- **CPU**: If you anticipate using map reducing, then the clock speed and the available processors become important considerations. Clock speed can also have a major impact on the overall performance when you are running a mongod with the majority of data in memory. In circumstances where you want to maximize the operations per second, you must consider including a CPU with a high clock/bus speed in your deployment strategy.
- **Replication** is used if high availability is one of the requirements. In any MongoDB deployment it should be standard to set up a replica set with at least three nodes.

  A 2x1 deployment is the most common configuration for replication with three nodes, where there are two nodes in one data center and a backup node in a secondary data center, as depicted in Figure 12-1.

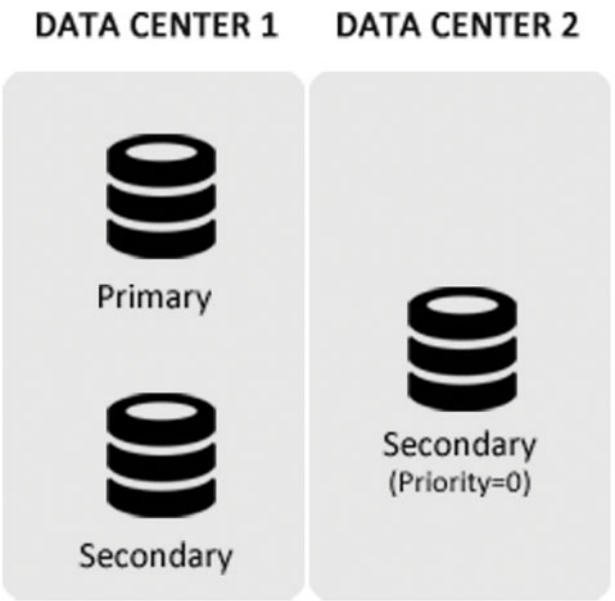

Figure 12-1. *MongoDB 2*1 deployment*

Hardware Suggestions from the MongoDB Site

The following are only intended to provide high-level guidance for a MongoDB deployment. The actual hardware configuration depends on your data, availability requirement, queries, performance criteria, and the selected hardware components' capabilities.

- **Memory**: Since memory is used extensively by MongoDB for a better performance, the more memory, the better the performance.
- **Storage**: MongoDB can use SSDs (solid state drives) or local attached storage. Since MongoDB's disk access patterns don't have sequential properties, SSDs usage can enable customers to experience substantial performance gains. Another benefit of using a SSD is if the working set no longer fits in memory, they provide a gentle degradation of performance.

  Most MongoDB deployments should use RAID-10.

  When using the WiredTiger storage engine, the use of a XFS file system is highly recommended due to performance issues.

  Also, do not use *huge pages* because MongoDB performs better with default virtual memory pages.
- **CPU**: Since MongoDB with a MMAPv1 storage engine rarely encounters workloads needing a large number of cores, it's preferable to use servers with a faster clock speed than the ones with multiple cores but slower clock speed. However, the WiredTiger storage engine is CPU bound, so using a server with multiple cores will offer a significant performance improvement.

Few Points to be Noted

To summarize this section, when choosing hardware for MongoDB, consider the following important points:

1. A faster CPU clock speed and more RAM are important for productivity.
2. Since MongoDB doesn't perform high amounts of computation, increasing the number of cores helps but does not provide a high level of marginal return when using the MMAPv1 storage engine.
3. Using SATA SSD and PCI (Peripheral Component Interconnect) provides good price/performance and good results.
4. It's more effective to spend on commodity SATA spinning drives.
5. MongoDB on NUMA Hardware: This point is only applicable for mongod running in Linux and not for instances that run on Windows or other Unix-like systems. NUMA (non-uniform memory access) and MongoDB don't work well together, so when running MongoDB on NUMA hardware, you need to disable NUMA for MongoDB and run with an interleave memory policy because NUMA causes a number of operational problems for MongoDB, including performance slowness for periods of time or high processor usage.

Coding

Once the hardware is acquired, consider the following tips when coding with the database:

- The first point is to think of the data model to be used for the given application requirement and to decide on embedding or referencing or a mix of both. For more on this, please look at Chapter tk. There's a trade-off between fast performance and guaranteed immediate consistency, so decide based on your application.
- Avoid application patterns that lead to unbounded growth of document size. In MongoDB, the maximum size for a BSON document is 16MB. Application patterns that make the documents grow in an unbounded way should be avoided.

  For instance, an application should not update documents a way that leads them to grow significantly. When the document size exceeds the allocated size, MongoDB will relocate the document. This process is not only time consuming, but is also resource intensive and can unnecessarily slow down other database operations. In addition, it can lead to inefficient use of storage.

  Note that the above mentioned limitation applies to the MMAPv1 storage engine. When using WiredTiger, the document is rewritten with every update.

For example, let's consider a blogging application. In this application, it's difficult to estimate how many responses a blog post will receive. The application is set to only display a subset of comments to the user, say the most recent comment or the first 10 comments. In this case, rather than creating an embedded model where the blog post and the user responses are maintained as a single document, you should create a referencing model where each response or group of responses are maintained as separate documents and then add a reference to the blog post in the documents. In this way, the unbound growth of the documents can be controlled, which will happen if you follow the first model of embedding the data.

- You can also design documents for the future. Although MongoDB provides the option of appending new fields within the documents as and when required, it has a drawback. When new fields are introduced, there might be a scenario where the document might not fit in the current space available, leading to MongoDB finding a new space for the document and moving it there, which might take time. So it is always efficient to create all the fields at the start if you are aware of the structure, irrespective of whether you have data available at that time or not. As highlighted above, the space will be allotted to the document and whenever value is there only needs to be updated. In doing so, MongoDB will not have to look for space; it merely updates the values entered, which is much faster.
- You can also create documents with the anticipated size wherever applicable. This point is also to ensure that enough space is allotted to the document and any further growth doesn't lead to hopping here and there for space.

This can be achieved by using a garbage field, which contains a string of the anticipated size while initially inserting the document and then immediately unsetting that field:

```
> mydbcol.insert({"_id" : ObjectID(..),......, "tempField" :
stringOfAnticipatedSize})
> mydbcol.update({"_id" : ...}, {"$unset" : {"tempField" : 1}})
```

- Subdocuments should always be used in a scenario when you know and will always know the names of the fields that you are accessing. Otherwise, use arrays.
- If you want to query for information that must be computed and is not explicitly present in the document, the best choice is to make the information explicit in the document. As MongoDB is designed to just store and retrieve the data, it does no computation. Any trivial computation is pushed to the client, leading to performance issues.
- Also, avoid `$Where` as much as possible because it's an extremely time- and resource-intensive operation.
- Use the correct data types while designing documents. For example, a number should be stored as a number data type only and not as a string data type. Using strings takes more space to store data and has an impact on the operations that can be performed on the data.
- Another thing to note is that strings in MongoDB are case sensitive. Hence a search for "practicalMongoDB" will not find "Practicalmongodb".

  Hence when doing a string search, you can do one of the following:

  - Store data in a normalized case format.
  - Use a regular expression with `/I` while searching.
  - Use `$toUpper` or `$toLower` in the aggregation framework.
- Using your own unique key as an `_id` will save a bit of space and will be useful if you are planning to index on the key. However, you need to keep the following things in mind when deciding to use your own key as `_id`:

  - You must ensure the uniqueness of the key.
  - Also, consider the insertion order for your key because the insertion order will identify how much RAM will be used to maintain this index.
- Retrieve fields as needed. When hundreds or thousands of requests are fulfilled per second, it's certainly advantageous to fetch only fields that are needed.
- Use GridFS only for storing data that is larger than what can fit in a single document or is too big to load at once on the client, such as videos. Anything that will be streamed to a client is a good candidate for GridFS.

- Use TTL to delete documents. If documents in a collection need to be deleted after a pre-defined time period, the TTL feature can be used to automatically delete the document after it reaches the predefined age.

Say you have a collection that maintains documents containing details of the user and the system interaction. The documents have a date field called **lastActivity**, which tracks the user and the system interaction. Let's say you have a requirement that says that you need to maintain the user session only for an hour. In this scenario, you can set the TTL to 3600 seconds for the field `lastActivity`. A background thread will run automatically and will check and delete documents that are idle for more than 3600 seconds.

- Use capped collections if you require high throughput based on insertion orders. In some scenarios, based on data size you need to maintain a rolling window of data in the system. For example, a capped collection can be used to store a high-volume system's log information to quickly retrieve the most recent log entries.
- Note that MongoDB's flexible schema can lead to inconsistent data if care is not taken. For example, the ability to duplicate data (embedded documents) if not updated properly can lead to data inconsistency, and so on. So it's very important to check for data consistency.
- Although MongoDB handles seamless failover, per good coding practice, the application should be well written to handle any exception and to gracefully handle such a situation.

Application Response Time Optimization

Once you start developing the application, one of the most important requirements is to have an acceptable response time. In other words, the application should respond instantly. You can use the following tips for optimization purposes:

- Avoid disk access and page faults as much as possible. Proactively figure out the data set size the application will be expected to deal with and add more memory in order to avoid page faults and disk read. Also, program your application in such a way that it mostly access data available in memory so page faults will happen infrequently.
- Create an index on the queried fields. If you create an index on the filter that you executing, the way the index is stored in memory will lead to less consumption of memory and hence will have a positive effect on the queries.
- Create covering indexes if the application involves queries that return a few fields as compared to the complete document structure.
- Having one compound index that can be used by maximum queries will also save on memory because instead of loading multiple indexes in memory, one index will suffice.
- Use trailing wildcards in regular expressions to reap the benefits from the associated index.

- Try to create indexes that reduce the possible documents to select from radically. An index on field "Gender" will not be as beneficial as an index on field "Phone Number."
- Indexing is not always good. You need to maintain an optimal balances of indexes used. Although you should create indexes for supporting your queries, you should also remember to delete indexes that are no longer used because every index has a cost associated for insert/update operations. If an index is not used but still it exists, it can have an adverse effect on the overall database capacity. This is especially important for insert-heavy workloads.
- Documents should be designed in a hierarchical fashion where related things are grouped together and are depicted as hierarchy wherever applicable. This will enable MongoDB to find the desired information without scanning the entire document.
- When applying an AND operator, you should always query from small resultset to a larger resultset because this will lead to querying a small number of documents. If you are aware of the most restrictive condition, that condition should go first.
- When querying with OR, you should move from a larger resultset to a smaller resultset because this will limit the search space for subsequent queries.
- The working set should fit in memory.
- Use the WiredTiger storage engine for write-heavy and I/O intensive applications due to its support of compression at the block as well as the index level.

Data Safety

You learned what you need to keep in mind when deciding on your deployment; you also learned a few important tips for good performance. Now let's look at some tips for data safety and consistency:

- Replication and journaling are two approaches that are provided for data safety. Generally it's recommended to run the production setup using replication rather than running it using a single server. And you should have at least one of the servers journaled. When it is not possible to have replication enabled and you are running on a single server, journaling provides data safety. Chapter tk explains how writes work when journaling is enabled.
- A repair should be the last resort for recovering data in the case of a server crash. Although the database might not be corrupted after running a repair, it will not contain all of the data.
- In a replicated environment, set `W` to the majority of safe writes. This is to ensure that the write is replicated to the majority of the members of the replica set. Although this will slow down the write operations, the write will be safe.
- Always specify `wtimeout` along with `w` when issuing the command in order to avoid the infinite waiting time.
- MongoDB should always be run in a trusted environment with rules to prevent access from all unknown systems, machines, or networks.

Administration

The following are some administration tips:

- Take instant-in-time backups of durable servers. To take a backup of a database with journaling enabled, you can take a file system snapshot or do a normal fsync+lock and then dump. Note that you can't just copy all of the files without fsync and locking because copying is not an instantaneous operation.
- Repair should be used to compact databases because it basically does a mongodump and then a mongorestore, making a clean copy of your data and, in the process, removing any empty "holes" in your data files.
- Database Profiler is provided by MongoDB. It logs fine-grained information on all the database operations. It can be enabled to either log information of all events or only events with durations exceeding a configurable threshold, which defaults to 100ms.

■ **Note** The profiler data is stored in a capped collection. As compared to parsing the log files, it may be easier to query this collection.

- An explain plan can be used to see how a query is being resolved. This involves information such as which index is used, how many documents are returned, whether the index is covering the query, how many index entries were scanned, and the time the query took to return results in milliseconds. When a query is resolved in less than 1ms, the explain plan shows 0. When you make a call to the explain plan, it discards the old plan and initiates the process of testing available indexes to ensure that the best possible plan is used.

Replication Lag

Replication lag is the primary administrative concern behind monitoring replica sets. Replication lag for a given secondary is the difference in time when an operation is written in primary and the time when the same was replicated on the secondary. Often, the replication lag remedies itself and is transient. However, if it remains high and continues to rise, there might be a problem with the system. You might end up either shutting down the system until the problem is resolved, or it might require manual intervention for reconciling the mismatch, or you might even end up running the system with outdated data.

The following command can be used to determine the current replication lag of the replica set:

```
testset:PRIMARY>rs.printSlaveReplicationInfo()
```

Further, you can use the `rs.printReplicationInfo()` command to fill in the missing piece:

```
testset:PRIMARY>rs.printReplicationInfo()
```

MongoDB Cloud Manager can also be used to view recent and historical replication lag information. The repl lag graph is available from the Status tab of each SECONDARY node.

Here are some tips to help reduce this time:

- In scenarios with a heavy write load, you should have a secondary as powerful as the primary node so that it can keep up with the primary and the writes can be applied on the secondary at the same rate. Also, you should have enough network bandwidth so that the ops can be retrieved from the primary at the same rate at which they are getting created.
- Adjust the application write concern.
- If the secondary is used for index builds, this can be planned to be done when there are low write activities on the primary.
- If the secondary is used for taking backups, consider taking backups without blocking.
- Check for replication errors. Run `rs.status()` and check the `errmsg` field. Additionally, the secondary's log files can be checked for any existing error messages.

Sharding

When the data no longer fits on one node, sharding can be used to ensure that the data is distributed evenly across the cluster and the operations are not affected due to resource constraints.

- Select a good shard key.
- You must use three config servers in production deployments to provide redundancy.
- Shard collections before they reach 256GB.

Monitoring

MongoDB system should be proactively monitored to detect unusual behaviors so that necessary actions can be taken to resolve issues. Several tools are available for monitoring the MongoDB deployment.

A free hosted monitoring service named MongoDB Cloud Manager is provided by MongoDB developers. MongoDB Cloud Manager offers a dashboard view of the entire cluster metrics. Alternatively, you can use nagios, SNMP, or munin to build your own tool.

MongoDB also provides several tools such as mongostat and mongotop to gain insights into the performance. When using monitoring services, the following should be watched closely:

- **Op counters**: Includes inserts, delete, reads, updates and cursor usage.
- **Resident memory**: An eye should always be kept on the allocated memory. This counter value should always be lower than the physical memory. If you run out of memory, you will experience slowness in the performance due to page faults and index misses.
- **Working set size**: The active working set should fit into memory for a good performance, so a close eye needs to be kept on the working set. You can either optimize the queries so that the working set fits inside the memory or increase the memory when the working set is expected to increase.

- **Queues**: Prior to the release of MongoDB 3.0, a reader-writer lock was used for simultaneous reads and exclusive access was used for writes. In such scenario, you might end up with queues behind a single writer, which may contain read/write queries. Queue metrics need to be monitored along with the lock percentage. If the queues and the lock percentage are trending upwards, that implies that you have contention within the database. Changing the operation to batch mode or changing the data model can have a significant, positive impact on the concurrency. Starting from Version 3.0, collection level locking (in the MMAPv1 storage engine) and document level locking (in the WiredTiger storage engine) have been introduced. This leads to an improvement in concurrency wherein no write lock with exclusive access will be required at the database level. So starting from this version you just need to measure the Queue metric.
- Whenever there's a hiccup in the application, the CRUD behavior, indexing patterns, and indexes can help you better understand the application's flow.
- It's recommended to run the entire performance test against a full-size database, such as the production database copy, because performance characteristic are often highlighted when dealing with the actual data. This also lets you to avoid unpleasant surprises that might crop up when dealing with the actual performance database.

Summary

In this chapter we provided various How To's to help you on journey with MongoDB.